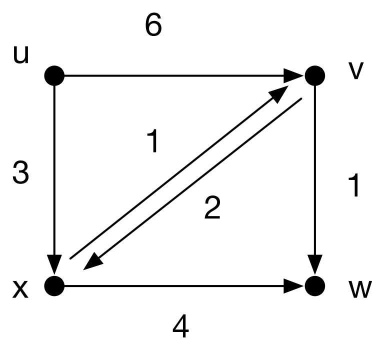
COMP 3711H – Fall 2014
Tutorial 7 – Solution Sketch

1. Execute Dijkstra's algorithm on the following digraph, where u is the source vertex.



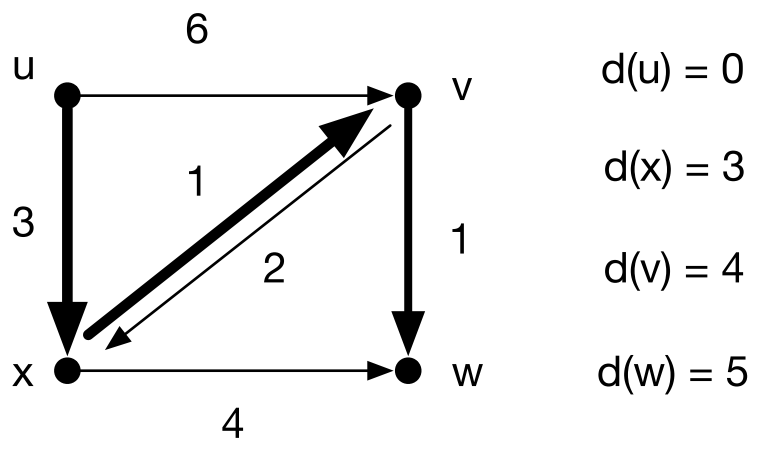
You need to indicate only the following:

(a) the order in which the vertices are removed from the priority queue.

(b) the final distance values $d[]$ for each vertex.

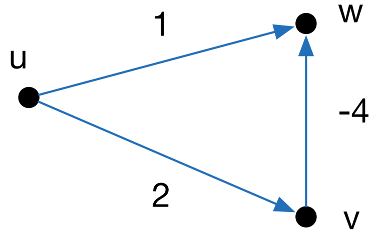
(c) the different distance values $d[]$ assigned to vertex b, as the algorithm executes.

Solution: (We only show the solution to (a) and (b).)



2. Give a simple example of a directed graph with negative-weight edges for which Dijkstra's algorithm produces incorrect answers. Why does the correctness proof of Dijkstra's algorithm not go through when negative-weight edges are allowed?

   Solution Consider the graph below with the algorithm starting at u. w is the next verex chosen and is given distance 1 from u. But, the real shortest distance from u to w is -2, by taking the path uvw.



   The correctness proof falls apart at various junctures. One of the fundamantal ones is that the proof assumed that a full path can not have shorter cost than any of its subaths but this is noolonger true when negative edges are allowed.

3. Suppose that instead of using a heap to store the tentative vertex distances, Dijkstra's algorithm just kept an array in which it stored each vertex's tentative distance. It then finds the next vertex by running through the array and choosing the vertex with lowest tentative distance. What would the algorithm's running time be? Is this better than our implementation for some graphs?

   Solution: Let m be number of vertices and m number of edges.

   The algorithm uses $O(n^2)$ time (independent of m). This is because at each step, the algorithm requires $O(n)$ time to scan through all of the remaining vertices. It also uses $O(m)$ time to update the tentative distances (since the cost of one update is $O(1)$ in changing an array value, rather than $O(\log n)$ for a decrease key).

   This is "better" than the $O((n+m)\log n)$ implementation we learned in class if the graph has $m = \Omega\left(\frac{n^2}{\log n}\right)$. For example, if the graph has $m = \Theta(n^2)$ then Dijkstra's original version runs in $O(n^2)$ time while the version taught in class uses $O(n^2 \log n)$ time.

4. Suppose that we are given a cable network of n sites connected by duplex communication channels. Unfortunately, the communication channels are not perfect. Each channel may fail with certain probability (also given in the input). The probabilities of failure may differ for different channels and they are mutually independent events. One of the n sites is the central station and your problem is to compute the most reliable paths from the central station to all other sites (i.e., the paths of lowest failure probabilities from the central station to all other sites). Design an algorithm for solving this problem, justify its correctness, and analyze its time and space complexities. Let $f(u, v)$ denote the failure probability for the channel between sites u and v.

   Solution. Let $f'(u, v) = 1 - f(u, v)$ which is the probability of edge (u, v) NOT failing.

The probabilty associated with a path $P = u_0, u_1, u_2, \ldots, u_t$ NOT failing is the product of the edge non-failures, i.e.,

$$Cost(P) = \prod_{i=0}^{t-1} f'(u_i, u_{i+1}).$$

Let u_0 be the central station Our goal is to find, for every v, a path from u_o to v with MAXIMAL cost.

If, instead of being the product of edge weights, the cost was the SUM of edge weights, and instead of wanting to find the Maximum we wanted to find the MINIMUM we would know how to solve the problem. This is just the single source shortest path problem which we know how to solve in $O((n+m)\log n)$ time using Dijkstra's algorithm.

Note that finding the maximum $Cost(P)$ is the same as finding the minimum $\frac{1}{Cost(P)}$. Also recall that the log of the product of values is the sum of the logarithms of the values.

This motivates setting $w(u,v) = -\log f'(u,v) = \log \frac{1}{f'(u,v)}$.

Next notice that

$$\log \frac{1}{Cost(P)} = \log \frac{1}{\prod_{i=0}^{t-1} f'(u_i, u_{i+1})} = \sum_{i=0}^{t-1} \log \frac{1}{f'(u_i, u_{i+1})} = \sum_{i=0}^{t-1} w(u,v)$$

Now finding a max-cost path is the same as finding a min $\frac{1}{Cost(P)}$ path which is the same as finding a min $\log(\frac{1}{Cost(P)})$ path. But, from the observation above, this is equivalent to finding a shortest *path using the weights $w(u,v)$. So, we can use Dijkstra's single source shortest path algorithm to solve the problem.*

One issue that we did not explicitly discuss is that Dijkstra's algorithm only works for edges that have non-negative weights. This is not a problem, though, because the $f'(u,v)$ are all probabilities between 0 and 1 Thus, the $w(u,v)$ are all non-negative.

5. Let G be a connected undirected graph with weights on the edges. Assume that all the edge weights are distinct. Prove that the edge with the smallest weight must be included in any minimum spanning tree of G. *You have to prove this from first principles, i.e., you are not allowed to use the Lemmas proven in class or assume the correctness of Kuskal's or Prim's algorithm.*

   Solution: Let e be the (unique) edge with the smallest weight and suppose that there exists some MST that does not contain e.

   Add e to T. Since T was a tree, adding e creastes a cycle. Let e' be any other edge on that cycle. Removing e' keeps the graph connected. Since it has $n-1$ edges (we added one edge and removed one edge) the resulting graph is also a tree. Call this new tree T'. The cost of T' is then

$$Cost((T') = Cost(T) - w(e') + w(e) < Cost(T)$$

   contradicting the optimality of T. Thus, e must have been in T to start with.